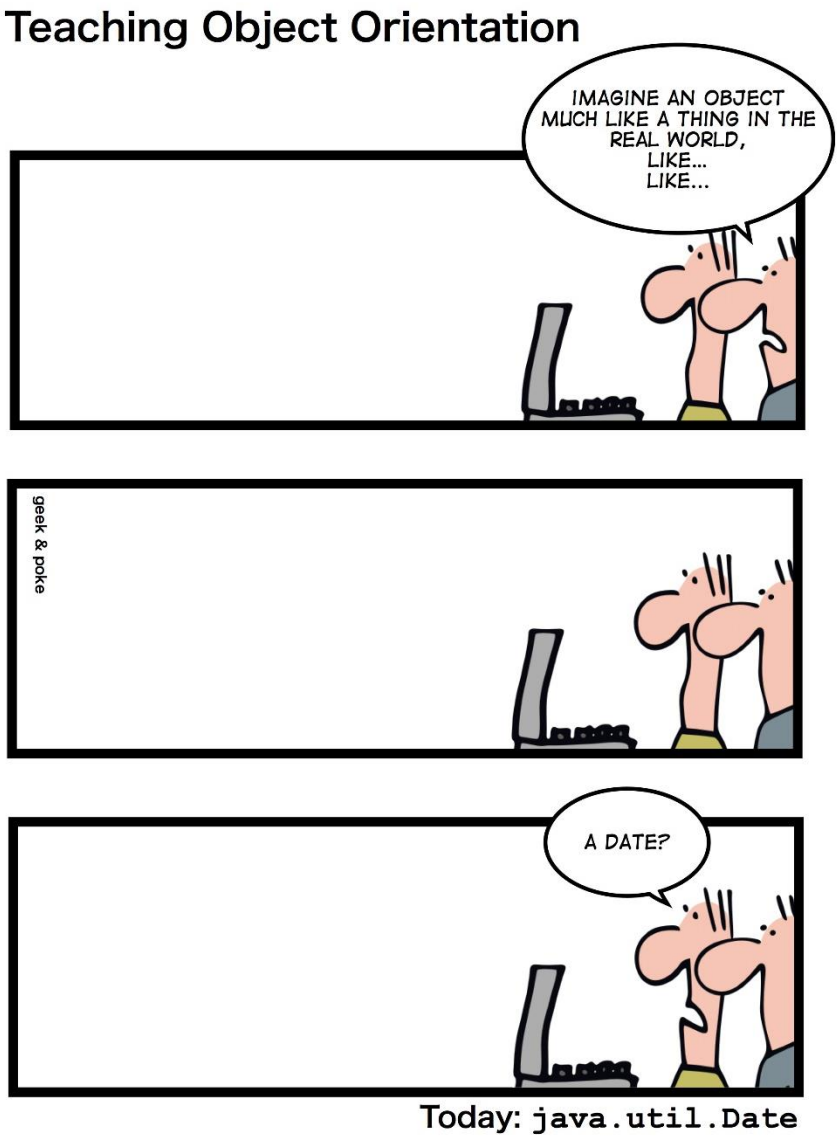
# Les objets

Antoine SCHMID, CFPT-Informatique
Avec la collaboration du CPLN

# Table des Matières

- Résumé de l’épisode précédent
- Les objets
- JavaScript et les objets
- Programmer avec les objets
- Les méthodes
- `this`
- Les objets prédéfinis de JavaScript



# Dans l’épisode précédent…

- ```
  function maFonction(param1, param2, ...) {
    return...
  }

  maFonction(arg1, arg2, ...);
  ```

- ```
  const maVariable = function(param1, param2) {
    return...
  }

  maVariable(arg1, arg2, ...);
  ```

- ```
  const maVarialbe = param1 => param1 + 12;

  maVariable(arg1);
  ```
  ```

# C’est quoi, un objet ?

- Un stylo…

- Quelles sont ses caractéristiques ?
  - Couleur (bleu, noir, rouge, …)
  - Type (à encre, à bille, à mine)
  - Marque (Mont blanc, Bic, Pelikan)

- Quelles sont ses fonctions ?
  - Permet d’écrire

# Et… ?

- Mais quel est le rapport avec la programmation ?

- La **programmation orientée objet** (en abrégé **POO**) est une manière d'écrire des programmes en utilisant des objets
- Quand on utilise la **POO**, on cherche à représenter le domaine étudié sous la forme d'objets informatiques.
- Chaque objet informatique modélisera un élément de ce domaine

# Objet informatique

- Un **objet informatique** est une entité qui possède

- Des **propriétés** : informations associées à l'objet
  - Couleur (bleu, noir, rouge, …)
  - Type (à encre, à bille, à mine)
  - Marque (Mont blanc, Bic, Pelikan)

- Des **méthodes** : actions que peut faire l'objet
  - Permet d'écrire

# JavaScript et les objets

- Comme de nombreux autres langages, JavaScript permet de programmer en utilisant des objets

- On dit que ce langage est **orienté objet**.

- Il vous fournit un certain nombre d'objets prédéfinis tout en vous permettant de créer les vôtres

# Création d'un objet

```
const stylo = {
  type: "bille",
  couleur: "bleu",
  marque: "Bic",
};
```

- Le code ci-dessus définit une variable nommée **stylo** dont la valeur est un objet : on dit aussi que **stylo** est un objet.
- Cet objet possède trois propriétés : **type**, **couleur** et **marque**.
- Chaque propriété possède **un nom et une valeur**, et est séparée des autres par une virgule (sauf la dernière).

# Accéder aux propriétés d’un objet

- Une fois l'objet créé, vous pouvez accéder à ses propriétés en utilisant la **notation pointée** : **`monObjet.maPropriete`**.

```
const stylo = {
  type: "bille",
  couleur: "bleu",
  marque: "Bic",
};

console.log(stylo.type); // "bille"
console.log(stylo.couleur); // "bleu"
console.log(stylo.marque); // "Bic"
```

# Modifier un objet

- Une fois un objet créé, on peut modifier les valeurs de ses propriétés avec la syntaxe **`monObjet.maPropriete = nouvelleValeur`**.

```
const stylo = {
  type: "bille",
  couleur: "bleu",
  marque: "Bic",
};

// Modification de la propriété "couleur"
stylo.couleur = "rouge";

// "J'écris avec un stylo bille rouge de marque Bic"
console.log(`J'écris avec un stylo ${stylo.type} ${stylo.couleur} de marque ${stylo.marque}`);
```

# Modifier un objet

- JavaScript offre même la possibilité d'**ajouter** dynamiquement de nouvelles propriétés à un objet déjà créé.

```
const stylo = {
  type: "bille",
  couleur: "bleu",
  marque: "Bic",
};

// Ajout de la propriété prix
stylo.prix = 2.5;

// "Mon stylo coûte 2.5 francs"
console.log(`Mon stylo coûte ${stylo.prix} francs`);
```

# Programmer avec les objets

- Essayons de programmer les bases d'un mini-jeu de rôle en utilisant des objets.

- Dans un jeu de rôle, chaque personnage est défini par de nombreuses caractéristiques…
  - Nom
  - Race
  - Santé
  - Force
  - Etc.

# Programmer avec les objets


# Création d’un personnage

- Je vous présente Aurora, le premier personnage de notre jeu

```
const heros = {
  nom: "Aurora",
  sante: 150,
  force: 25
};
```

# Création d’un personnage

```
const heros = {
  nom: "Aurora",
  sante: 150,
  force: 25
};
```

- Aurora est représentée par un objet **heros** ayant trois propriétés
  - **nom**
  - **sante**
  - **force**

- Elles ont chacune une valeur et, ensemble, définissent l'état de notre personnage à un instant donné.

# Création d'un personnage

```
const heros = {
  nom: "Aurora",
  sante: 150,
  force: 25
};
```

- Aurora s'apprête à vivre une longue série d'aventures, dont certaines pourront modifier ses caractéristiques

```
// "Aurora a 150 points de vie et 25 en force"
console.log(`${heros.nom} a ${heros.sante} points de vie et ${heros.force} en force`);

console.log(`${heros.nome} est blessée par une flèche`);
heros.sante -= 20;

console.log(`${heros.nome} trouve un bracelet de force`);
heros.force += 10;

// "Aurora a 130 points de vie et 35 en force"
console.log(`${heros.nom} a ${heros.sante} points de vie et ${heros.force} en force`);
```

# La notion de méthode

- Dans notre programme, on remarque que l'instruction **console.log(...)** qui affiche les caractéristiques du personnage est dupliquée

- Ce qui est mal ! (DRY)

# Ajout d’une méthode à un objet

- Premier essai : on crée une fonction pour éviter le doublon…

```
// Renvoie la description d’un personnage
function decrire(personnage) {
  return `${personnage.nom} a ${personnage.sante} points de vie et ...`;
}

// "Aurora a 150 points de vie et 25 en force"
console.log(decrire(heros));
```

- La fonction **`decrire()`** n'utilise que les propriétés d'un personnage, qui est passé en paramètre

# Ajout d’une méthode à un objet

- Plutôt que de la définir de manière externe, nous pouvons l'ajouter à la définition de notre objet sous la forme d'une nouvelle propriété dont la valeur est une fonction.

```
const heros = {
  nom: "Aurora",
  sante: 150,
  force: 25,

  // Renvoie la description du personnage
  decrire() {
    return `${this.nom} a ${this.sante} points de vie et ${this.force} en force`;
  }
};

// "Aurora a 150 points de vie et 25 en force"
console.log(heros.decrire());
```

# Ajout d’une méthode à un objet

- Une propriété dont la valeur est une fonction est appelée une **méthode**.

- Une méthode permet de définir une action pour un objet.

- On dit également qu'une méthode ajoute à cet objet un comportement.

# Appel d’une méthode sur un objet

```
// "Aurora a 150 points de vie et 25 en force"
console.log(heros.decrire());
```

- **`decrire(aurora)`** appelle la fonction **`decrire()`** en lui passant l'objet **perso** en paramètre. Dans ce cas, la fonction est externe à l'objet.
- **`heros.decrire()`** appelle la fonction **`decrire()`** sur l'objet **`heros`** . Dans ce cas, la fonction fait partie de la définition de l'objet : il s'agit d'une **méthode**.

# Le mot-clé `this`

```
const heros = {
  ...

  // Renvoie la description du personnage
  decrire() {
    return `${this.nom} a ${this.sante} points de vie et ${this.force} en force`;
  }
};
```

- **`this`** est défini automatiquement par JavaScript à l'intérieur d'une méthode et représente l'objet sur lequel la méthode a été appelée
- La méthode **`decrire()`** ne prend plus de personnage en paramètre : elle utilise **`this`** pour accéder aux propriétés de l'objet sur lequel elle a été appelée.

# Les objets prédéfinis de JavaScript

- Le langage JavaScript met à la disposition des programmeurs un certain nombre d'objets standards qui peuvent rendre de multiples services

- L'objet **`console`** donne accès à la console pour y écrire des messages texte.
- L'objet **`Math`** rassemble des fonctionnalités mathématiques. Par exemple, **`Math.random()`** renvoie un nombre aléatoire entre 0 et 1.

# Questions

# Exercices

- Exercices 7.x : **Objets**